January 2008

A new Kalman-filter-based framework for fast and accurate visual tracking of rigid objects

Yoon Youngrock

A. Kosaka

A. C. Kak

Youngrock, Yoon; Kosaka, A.; and Kak, A. C., "A new Kalman-filter-based framework for fast and accurate visual tracking of rigid objects" (2008). *Department of Electrical and Computer Engineering Faculty Publications.* Paper 53.
http://dx.doi.org/http://dx.doi.org/10.1109/TRO.2008.2003281

A New Kalman-Filter-Based Framework for Fast and Accurate Visual Tracking of Rigid Objects

Youngrock Yoon, *Member, IEEE*, Akio Kosaka, *Member, IEEE*, and Avinash C. Kak

***Abstract*—The best of Kalman-filter-based frameworks reported in the literature for rigid object tracking work well only if the object motions are smooth (which allows for tight uncertainty bounds to be used for where to look for the object features to be tracked). In this contribution, we present a new Kalman-filter-based framework that carries out fast and accurate rigid object tracking even when the object motions are large and jerky. The new framework has several novel features, the most significant of which is as follows: the traditional backtracking consists of undoing one-at-a-time the model-to-scene matchings as the pose-acceptance criterion is violated. In our new framework, once a violation of the pose-acceptance criterion is detected, we seek the best largest subset of the candidate scene features that fulfill the criterion, and then continue the search until all the model features have been paired up with their scene correspondents (while, of course, allowing for nil-mapping for some of the model features). With the new backtracking framework, our Kalman filter is able to update on a real-time basis the pose of a typical industrial 3-D object moving at the rate of approximately 5 cm/s (typical for automobile assembly lines) using off-the-shelf PC hardware. Pose updating occurs at the rate of 7 frames per second and is immune to large jerks introduced manually as the object is in motion. The objects are tracked with an average translational accuracy of 4.8 mm and the average rotational accuracy of 0.27°.**

***Index Terms*—3-D pose estimation, extended Kalman Filter (EKF), object tracking, visual servoing.**

I. Introduction

OBJECT tracking has numerous applications such as traffic surveillance [1]–[4], augmented reality [5], mobile robot navigation [6], robotic assembly on a moving line [7], etc. For many of these applications involving 3-D objects, it is not sufficient to just do 2-D tracking; the tracking algorithm must also provide the 3-D pose of the object. For example, for the case of robotic assembly on a moving line in a modern factory, it is essential that the 3-D pose of the object being tracked—such as a car engine cover—be fully known at all times so that the robot end-effector can interact with the object in meaningful ways. Since the 3-D pose of a rigid object involves 6 DOF, three for translation and three for rotation, the tracking algorithm for such applications must yield all six parameters of the pose. These parameters must obviously be estimated despite occlusions, background clutter, varying illumination, etc.

The contributions that have been made in the past on tracking that allow for the estimation of the 3-D pose of an object fall into two categories depending on whether or not backtracking is used in matching model and scene features. In the first category, we have approaches that use point features. The matching strategies used in this category are usually one-shot, meaning the scene features are paired up with the best possible candidates from the model (but this is done only once), and iterative in pose space, meaning a gradient-based approach is used to find the best possible 3-D pose that minimizes some error functional between the model and the scene. This synopsis applies to the work reported in [8]–[10].

With such a one-shot correspondence search strategy, the pose estimate often drifts away from the true pose, especially when the target object moves nonsmoothly, and the predicted pose for each frame has large discrepancy from the true pose. In order to alleviate this problem, some approaches employ a robust estimator, such as the M-estimator [9], for minimizing the error function, or a voting-based strategy, such as the generalized Hough transform in the pose space [1]. Marchand *et al.* [11] estimate a rough location of the target in the scene by calculating the 2-D affine transformation between each consecutive frame, and then, applying multiresolution generalized Hough transform to estimate the finer pose. Vacchetti *et al.* [12] use an appearance-based offline registration method to get around the drift problem associated with the one-shot approaches to pose estimation. Recently, there have been attempts to get around the need for explicit matches between the model and the scene by directly estimating the location of model contours in the scene [4].

These approaches are not suitable for accurate estimation of 3-D pose on a continuing basis as an object is tracked against cluttered backgrounds. The main source of difficulty with these approaches appears to be a lack of a backtracking-based search framework for matching model features with scene features. A backtracking-based solution to the problem must of necessity include some sort of an uncertainty model for locational and other properties of scene features. We believe that the problems that can be caused by the lack of backtracking also apply to the recent work of Lippiello *et al.* [13].

The second category of approaches for tracking while the 3-D pose is constantly updated combines a backtracking-based strategy for matching with pose-uncertainty modeling in order

Manuscript received February 24, 2007; revised September 27, 2007 and February 14, 2008. First published September 23, 2008; current version published October 31, 2008. This paper was recommended for publication by Associate Editor D. Sun and Editor L. Parker upon evaluation of the reviewers' comments. This work was supported by Ford Motor Company.

Y. Yoon was with the Robot Vision Laboratory, Purdue University, West Lafayette, IN 47907 USA. He is now with Tandent Vision Science, Inc., Knoxville, TN 37921 USA (e-mail: yoony@ecn.purdue.edu).

A. Kosaka is with the Robot Vision Laboratory, Purdue University, West Lafayette, IN 47907 USA, and also with the Future Creation Laboratory, Olympus Corporation, Tokyo 163-0914, Japan (e-mail: kosaka@ecn.purdue.edu).

A. C. Kak is with the Robot Vision Laboratory, Purdue University, West Lafayette, IN 47907 USA (e-mail: kak@ecn.purdue.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2008.2003281

to achieve greater robustness in matching [2], [6], [14], [15]. The contribution by Lowe [14] uses the Gauss–Newton method for minimizing the error between the predicted pose and the true pose. Koller *et al.* [2] and Tonko and Nagel [15] use the extended Kalman filter (EKF) for updating the positional uncertainties associated with the model features. Although these approaches are similar to ours in using EKF for estimating the target object pose, the feature-correspondence-seeking strategies in these approaches are not as elaborate as needed to accommodate jerky motions of the sort we address in this paper. The backtracking strategy used in [2] is similar to that described in [6]. Such one-feature-at-a-time backtracking often fails when the motion is too jerky, as we will argue in the rest of this section. And the EKF implementation described in [15] does not even use any backtracking. So, it too cannot be expected to deal with sudden large changes in object pose during tracking. As we will explain in this paper, backtracking is necessary for coping with large sudden variations in object pose, but the strategy used for backtracking must allow the system to completely abandon a pose hypothesis as opposed to merely undoing a previous model-to-scene match for a single feature.

Of the approaches listed above, the prior contribution by Kosaka and Kak [6] is particularly relevant to the new research reported here. Although this Kalman-filter-based formalism was originally developed for vision-based mobile robot navigation, it was later shown to be useful for 3-D object tracking also [16]. The work reported in both [6] and [16] is based on an incremental pose-update scheme in a prediction–verification framework. In this framework, the pose of an object in each input scene is predicted with uncertainty. As the features in the model of the object are sequentially matched with the features in the input scene, an EKF is used to reduce the pose uncertainty by observing the error between the matched features. As more and more features are matched, the estimation of the target pose becomes increasingly accurate.

The goal in this paper is to use the work reported in [6] as a starting point for developing a fast and accurate 3-D tracker that also continuously yields the 3-D pose of the object being tracked even when the object motions are large and jerky. Our research here goes beyond what was reported in [6] in the following important ways.

1) When a target object moves with a large variation in its motion, the predicted statistics of the object pose for each image frame tend to deviate significantly from the true pose. To solve this problem, a large amount of motion uncertainty has to be assigned to the predicted pose. As our experiments have shown, large uncertainty in the predicted pose causes the maximum likelihood frameworks for feature correspondence estimation, such as the one in [6], to break down. To understand what we mean by "break down," note that all that a Kalman filter does is to update the pose mean and covariance. The uncertainty associated with such updates will always be smaller with each iteration even when we use inappropriate matchings between the model features and the scene features. *Inappropriate pairings between the model and the scene features are more likely to take place in the presence of large motion uncertainties.* To get around this problem in the research reported here, after we have updated the pose, we reexamine the model-to-scene feature pairings that went into the update calculations. If the new pose (and the new bounds on the uncertainties) does not support these pairings, they are undone in their entirety (*as opposed to one-at-a-time in traditional implementation of the backtracking step in EKF [6]*) and new pairings sought. This process is repeated until the updated pose and the set of matched model-to-scene features support each other fully and reciprocally. Detailed description of the hypothesis generation and verification scheme is presented in Sections III-F and III-G.
2) While more robust, being iterative, the framework mentioned earlier can extract a performance penalty unless care is taken in the initial selection of model-to-scene feature matchings. To minimize this potential performance penalty, our system first rank orders the model features on the basis of a number of criteria. At each iteration, scene features are sought for only the top-ranked model features. Experiments have shown that this significantly reduces the number of backtrackings needed in our framework. Rank ordering of the model features is described in Section III-F.

In the next section, we present an overview of our tracking system. In Sections III, IV and V, we present detailed description of our pose estimation algorithm that is used iteratively in the tracking system. In Section VI, we present pose estimation and tracking results with a few target objects. Finally, we conclude in Section VII.

II. Tracking Algorithm

A. Workspace Description and Definition of Pose

We use three coordinate frames to represent features in our workspace: the world coordinate frame, the camera coordinate frame, and the target object coordinate frame. The world coordinate frame, which we denote as W, is the reference coordinate frame for points in the workspace. This frame is usually attached to a fixed reference in the workspace. The camera coordinate frame C is the camera-centered coordinate frame whose x and y axes are aligned with horizontal and vertical directions of the camera image plane, respectively, and z axis is aligned perpendicular to the image plane. The target coordinate frame T is a coordinate frame that all model feature points of a target object are defined with respect to. Fig. 1 shows these coordinate frames and how they are related to each other.

The transformation of the feature vectors from T to C has 6 DOF: three for translation and three for rotation. We define the pose of an object as the 6-D random vector $\mathbf{p} = (t_x, t_y, t_z, \phi_x, \phi_y, \phi_z)^T$, where t_x, t_y, t_z are the translational components and ϕ_x, ϕ_y, ϕ_z are the rotational components of the transformation from T to C. The three rotational components represent the Euler-III-type angles of rotation about the three axes x, y, z of C, respectively, as defined in [17]. The use of boldface font for $\mathbf{p}$ signifies that $\mathbf{p}$ is a random vector. We assume $\mathbf{p}$ has Gaussian distribution with mean $\bar{\mathbf{p}}$ and covariance

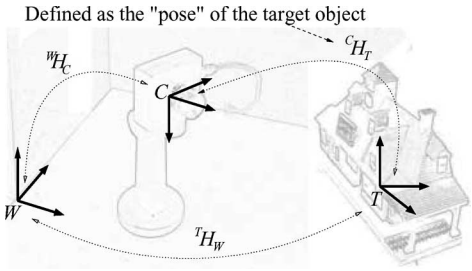


Fig. 1. Tracking workspace definition.

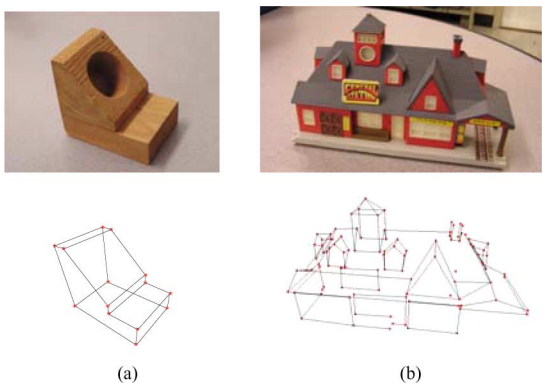


Fig. 2. Example of rigid objects and their wireframe models. (a) A wood block. (b) A train station object.

matrix $\Sigma_{\mathbf{p}}$. Alternatively (and more usefully), the pose vector $\mathbf{p}$ is represented in the form of a homogeneous transformation matrix from T to C and denoted as ${}^{C}H_{T}$ using the Denavit–Hartenberg notation [18]. When we want to show that the elements of this matrix are directly related to the pose vector, we write the matrix as ${}^{C}H_{T}(\mathbf{p})$.

Although we formulate our pose estimation algorithm for the relative pose of T with respect to C, the algorithm can be easily adapted to estimate the pose with respect to W by replacing ${}^{C}H_{T}(\mathbf{p})$ in (1) in the next section with ${}^{C}H_{W}\,{}^{W}H_{T}(\mathbf{p})$, where ${}^{W}H_{T}(\mathbf{p})$ represents the pose of the target with respect to W. ${}^{C}H_{W}$ is given by camera calibration.

B. Modeling Target Objects

The model features extracted from the wireframe model, meaning the actual straight-line edges on the boundary surface of the object, are represented by the Cartesian coordinates of the two extremities in T. That is, a model feature m is represented by two 3-D vectors $m^{k} = (x^{k}, y^{k}, z^{k})^{T}, k = 1, 2$ that are the 3-D Cartesian coordinates of the two extremities of m in the coordinate frame T. The superscript k of each vector denotes the extremity that it represents.

Fig. 2(a) shows a simple mostly polyhedral object at the top and its wireframe model. Fig. 2(b) shows a more complex object at the top and its wireframe model at the bottom. We refer to the latter object as the train station object. This object will be used to illustrate the various steps of our tracking algorithm in the rest of this paper.

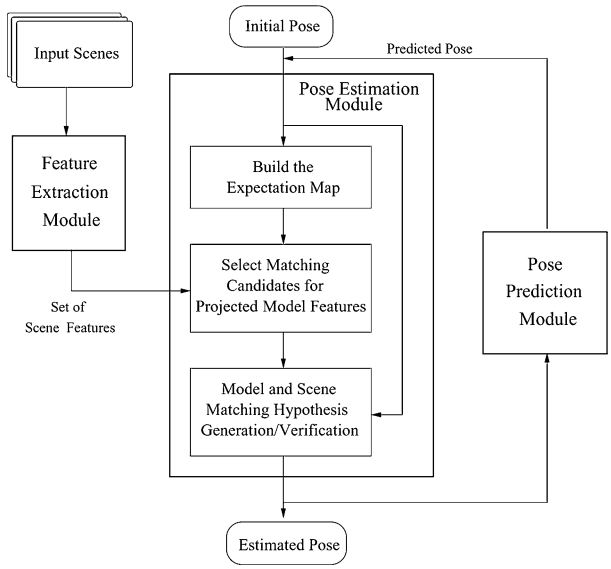


Fig. 3. Overview of our visual tracking algorithm.

C. Tracking System Overview

Tracking in our system is executed by applying a model-based pose estimation algorithm to each consecutive frame in the input image sequence. An overview of our tracking system is depicted in Fig. 3. As shown in this figure, our tracking system consists of three modules: the feature extraction module, the pose estimation module, and the pose prediction module.

The feature extraction module extracts straight-line feature descriptors from the input scene image along with the associated measurement uncertainties. This module is presented in Section IV.

The pose estimation module searches for the best match between the features received from the feature extraction module and the model features projected into the camera image by the pose prediction module. The pose estimation module uses an EKF to estimate the pose in such a way as to minimize the error in the image space between the object as perceived through the extracted features and the object model as projected into the camera image. Details of this module are presented in Section III.

The pose prediction module predicts the pose of the target for the next image frame using a linear extrapolation method based on motion estimates of the target. Such a predicted pose of the target is used to project the object model into the camera image for constructing the expected view of the object. The initial value of the target object pose in the beginning of the tracking sequence is assumed to be given.[1] Details of this module are presented in Section V.

[1]For the visual servoing experiment that is described in Section VI-D, the initial pose of the target is provided by a coarse-control object tracking module in our distributed control architecture. Detailed descriptions of our control architecture and the trackers are presented in [19]. For non-robotic tracking experiments, such as when tracking a hand-held object with a stationary or moving camera, the initial pose is assumed to be given by human users. This can easily be done with an appropriate GUI that allows a human to translate and rotate the projected image of the model object until it coincides with the actual camera image of the object.

III. Pose Estimation Module

Although feature extraction is the first step that a camera image is subjected to, we will go ahead and explain how we carry out pose estimation in our system. The discussion on pose estimation will permit us to explain the overall uncertainty calculus used by our system, which will subsequently result in a more efficient explanation for the feature extraction module. Representing and manipulating scene and model uncertainties are key aspects of our pose estimation algorithm.

Overall, our system is aware of two different kinds of uncertainties and it keeps track of them separately: the feature extraction module associates a *measurement uncertainty* with each straight-line feature. The measurement uncertainty depends on what it takes to group together a series of edge fragments into a single straight-line feature. The other kind of uncertainty—the kind that is the focus of this section—is due to the discrepancy between the true pose of the model object and its currently known pose as the object is in motion. This is the uncertainty that must be associated with the model features that are projected into the camera image by the pose prediction module. We will refer to this uncertainty as the *pose prediction uncertainty*.

For describing our pose estimation algorithm in detail, we start with presenting the definition of the image error between the projected model features and their corresponding scene features in the following section.

A. Constraint Equation for Pose Error in Image Space

Previously, we talked about a model object as being defined in an object-centered coordinate frame denoted as T. As an object moves in space, this coordinate frame moves with the object. In other words, the pose of a moving target object in its own coordinate frame T never changes. Our goal in tracking is to constantly update the pose vector $\mathbf{p}$, which is equivalent to updating the transformation matrix ${}^{C}H_{T}(\mathbf{p})$. The pose of an object is estimated by predicting the current value and uncertainty of the ${}^{C}H_{T}(\mathbf{p})$ matrix from its previous value and the motion uncertainty parameters currently in effect, and then, using this predicted matrix to project the relevant model features into the camera frame C. For obvious reasons, we can refer to these projected model features in the camera frame as the expectation map. A difference between the expectation map and what the camera sees at the current moment, the difference being caused by the pose error, is then used for updating the $\mathbf{p}$ vector, the uncertainties, and the various motion parameters. Since ${}^{C}H_{T}(\mathbf{p})$ is random, the locations of the projected model features are denoted by their means and covariances that can be derived from the mean and covariance of $\mathbf{p}$ at the time of the projection.

The expectation map for a given value of $\mathbf{p}$ is constructed by applying ${}^{C}H_{T}(\mathbf{p})$ to the vectors representing the end points of the straight-line model features. For such a projection, we use a perspective projection model that is widely used for such purposes [20]. Using the perspective projection model, the projection of the two end points m_i^k, $k = 1, 2$ of a model feature m_i is described by the following equation:

$$\begin{pmatrix} u_{m_i}^k w \\ v_{m_i}^k w \\ w \end{pmatrix} = \begin{pmatrix} \alpha_u & 0 & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} {}^{C}H_{T}(\mathbf{p}) \begin{pmatrix} x_i^k \\ y_i^k \\ z_i^k \\ 1 \end{pmatrix} \quad (1)$$

where $u_{m_i}^k, v_{m_i}^k$ are the image coordinates, w the scaling parameter, and (x_i^k, y_i^k, z_i^k) the actual coordinates in the coordinate frame T for the end points m_i^k. $\alpha_u, \alpha_v, u_0, v_0$ are the intrinsic camera parameters that are given by the camera calibration, for which we use the algorithm presented in [21].[2] Using this equation, we denote the projection of a model feature m_i for a given value of $\mathbf{p}$ as a 4-D vector $g_{m_i,\mathbf{p}}$ as follows:

$$g_{m_i,\mathbf{p}} = [u_{m_i}^1, v_{m_i}^1, u_{m_i}^2, v_{m_i}^2]^T. \quad (2)$$

For a given distribution of $\mathbf{p}$ with mean $\bar{\mathbf{p}}$ and covariance matrix $\Sigma_{\mathbf{p}}$, the mean of $g_{m_i,\mathbf{p}}$, which we denote as $\bar{g}_{m_i,\mathbf{p}}$, is calculated by replacing ${}^{C}H_{T}(\mathbf{p})$ with ${}^{C}H_{T}(\bar{\mathbf{p}})$ in (1). The uncertainty of $g_{m_i,\mathbf{p}}$, which we denote as $\Sigma_{g_{m_i,\mathbf{p}}}$, is approximated using $\Sigma_{\mathbf{p}}$ as follows:

$$\Sigma_{g_{m_i,\mathbf{p}}} = J(g_{m_i,\mathbf{p}}, \mathbf{p}) \Sigma_{\mathbf{p}} J(g_{m_i,\mathbf{p}}, \mathbf{p})^T \quad (3)$$

where $J(g_{m_i,\mathbf{p}}, \mathbf{p})$ is the Jacobian matrix of the pixel coordinates of $g_{m_i,\mathbf{p}}$ with respect to $\mathbf{p}$. For estimating the pose of the object, the projected model features in the expectation map must be matched with the straight-line features that are extracted from the edge map of the input scene. Let z_j be the scene feature that is selected for matching with the camera projection of the model feature m_i. We denote this scene feature as a 4-D vector with the image coordinates of its two end points as follows:

$$z_j = [u_{z_j}^1, v_{z_j}^1, u_{z_j}^2, v_{z_j}^2]^T. \quad (4)$$

Because of the various uncertainties that are involved in edge detection and straight-line extraction, z_j is also a random vector. We assume that this vector can also be characterized by a Gaussian distribution. Details on estimating the mean and covariance for z_j are presented later in Section IV when we describe our algorithm for extracting such scene features from input images.

The vector $g_{m_i,\mathbf{p}}$ gives us the predicted locations of the end points of the model straight-line feature m_i. On the other hand, the vector z_j corresponds to the actual measured locations of such end points in the image space. If the predicted pose corresponds exactly to the current pose of the target, then obviously, $g_{m_i,\mathbf{p}} - z_j$ will be zero. So, when that is not the case, any differences between the two must be minimized. Therefore, the following constraint equation must be satisfied by any pose update mechanism:

$$f(\mathbf{p}, m_i, z_j) = g_{m_i,\mathbf{p}} - z_j = 0. \quad (5)$$

[2]For this projection equation, we assume that the input image is free of any lens distortion effect. We use the lens distortion parameters that are estimated by the algorithm in [21] to remove the distortion from the input images. The software routines used for such distortion removal were drawn from the *OpenCV* library [22].

B. EKF-Based Recursive Pose Update Framework

We use a recursive framework that is similar to the framework used in [6] for updating the pose of the target given the error between the model features and the corresponding scene features. For each model and scene feature pair in a given set of feature correspondences, our framework uses an EKF [23] to transform the pose parameters to presumably more accurate pose parameters that optimally minimize the error between the corresponding features. The updated pose parameters serve as the initial state for the next pose update with another feature correspondence. The fact that the updated pose parameters are used as the initial state for the next update explains why we call our framework recursive.

Let $C = \{(m_1, z_{j_1}), \ldots, (m_{N_C}, z_{j_{N_C}})\}$ be a set of model and scene feature correspondences where (m_i, z_{j_i}), $i = 1, \ldots, N_C$ denotes that the model feature m_i is matched with the scene feature z_{j_i}. N_C is the cardinality of C. We also denote the pose vector after pose update using the match (m_i, z_{j_i}) as $\mathbf{p_i}$ and its corresponding mean and covariance as $\bar{\mathbf{p}}_\mathbf{i}$ and $\Sigma_{\mathbf{p_i}}$, respectively. Our pose update equations transform $\mathbf{p_{i-1}}$ into $\mathbf{p_i}$ while minimizing the error between z_{j_i} and $g_{m_i,\mathbf{p_{i-1}}}$. With regard to the pose update processing for each new image frame, the statistics of the initial pose $\mathbf{p_0}$ are given by the pose prediction module using the estimated pose from the previous image frame as described in Section V. Let $\hat{z}_{j_i}$ represent the actual measured z_{j_i}. We assume that the feature measurement error is additive white Gaussian and we denote this error as ξ_{j_i} with error covariance V_{j_i}. By linearizing and rearranging (5) in the vicinity of $\bar{\mathbf{p}}_\mathbf{i-1}$ and $\hat{z}_{j_i}$ using the Taylor's series expansion, we get the following equation:

$$y_i = M_i \mathbf{p} + e_{j_i} \tag{6}$$

where

$$\begin{aligned} y_i &= -f(\bar{\mathbf{p}}_\mathbf{i-1}, m_i, \hat{z}_{j_i}) + \frac{\partial f(\mathbf{p}, m_i, z_{j_i})}{\partial \mathbf{p}} \bar{\mathbf{p}}_\mathbf{i-1} \\ M_i &= \frac{\partial f(\mathbf{p}, m_i, z_{j_i})}{\partial \mathbf{p}} \\ e_{j_i} &= \frac{\partial f(\mathbf{p}, m_i, z_{j_i})}{\partial z_{j_i}} (z_{j_i} - \hat{z}_{j_i}). \end{aligned} \tag{7}$$

We denote the covariance matrix of e_{j_i} as E_{j_i}, which can be easily calculated from the covariance matrix V_{j_i} of ξ_{j_i}.

Using the EKF theory, the minimization of the constraint in (5) via the linearizations in (6) through (7) is achieved if the statistics of the state vector $\mathbf{p_i}$ are updated by the following equations:

$$\begin{aligned} \bar{\mathbf{p}}_\mathbf{i} &= \bar{\mathbf{p}}_\mathbf{i-1} - K_i f(\bar{\mathbf{p}}_\mathbf{i-1}, m_i, \hat{z}_{j_i}) \\ K_i &= \Sigma_{\mathbf{p_{i-1}}} M_i^T (E_{j_i} + M_i \Sigma_{\mathbf{p_{i-1}}} M_i^T)^{-1} \\ \Sigma_{\mathbf{p_i}} &= (I - K_i M_i) \Sigma_{\mathbf{p_{i-1}}}. \end{aligned} \tag{8}$$

C. Building the Expectation Map

For constructing the expectation map, we identify two groups of model straight-line features that should not be projected onto the camera image plane. The first is the group of model features that are self-occluded by other parts of the object for a given pose matrix. For identifying this type of model features, we use the binary space partitioning (BSP) tree representation of a polyhedral model [24].

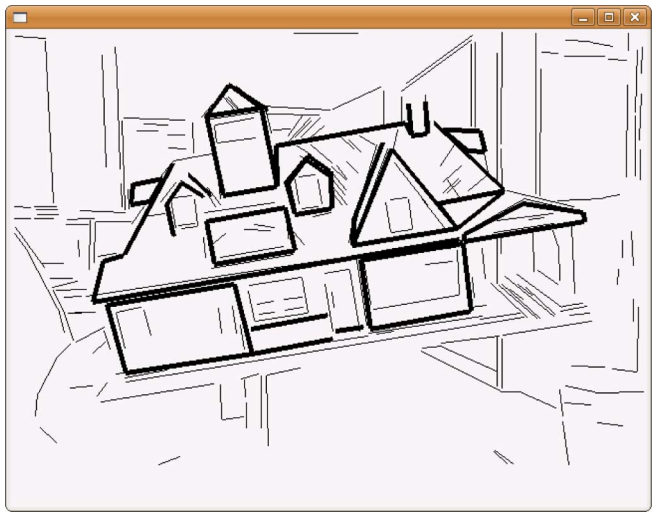


Fig. 4. For the pose of the object shown, the expectation map as derived from the 3-D object model consists of the thick black lines in the figure. The thin black lines shown in the figure constitute the scene features extracted from the image.

The second group is the group of model straight-line features that are parallel to the optic axis of the camera. Although such a group of line segments is expected to be visible in the expectation map, it is viewed as a group of very short line segments or points. It is obviously undesirable to match these kinds of model features to the scene. Currently, we exclude the line segments whose direction is within 20° of the optic axis of the camera. Fig. 4 shows an example of an expectation map constructed for the target object of Fig. 2(b). This map is superimposed on the scene edges extracted from an image frame. The expectation map consists of thick black lines.

D. Selecting Match Candidates for Projected Model Features

The locations of the projected model features in the image space possess uncertainty owing to the uncertainty associated with the predicted pose of the target. This positional uncertainty for the projected model features defines regions in the image space in which the system should search for the scene feature candidates to be matched with the projected model features.

For each projected model edge $g_{m_i,\mathbf{p_0}}$, the covariance matrices for the positions of its two end points are the 2×2 submatrices in the diagonal of $\Sigma_{g_{m_i,\mathbf{p_0}}}$ that is calculated by (3). These covariance matrices define elliptical regions around the end points of $g_{m_i,\mathbf{p_0}}$ in the image space. We define an approximate convex hull that encloses these elliptical regions and use this convex hull as the search region for match candidates. Fig. 5 shows an example of defining the search region for a projected model feature in the image space. Note that our convex hull is approximate in the sense that it is polygonal, which allows for efficient computations.

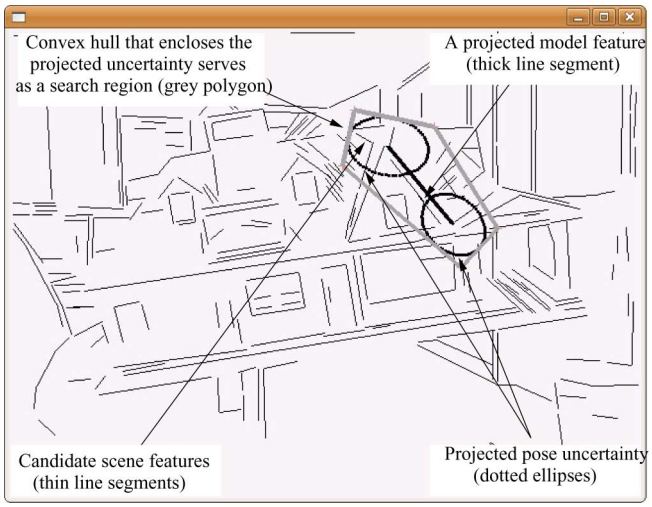


Fig. 5. Search region for candidate scene features.

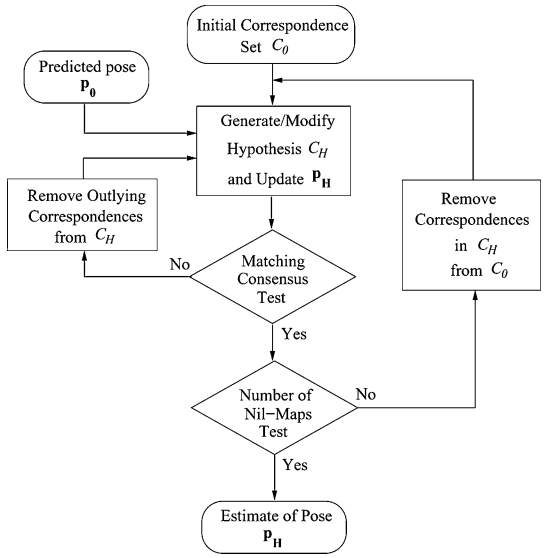


Fig. 6. Overview of the hypothesis generation and verification scheme.

For the search region defined around $g_{m_i,\mathbf{p_0}}$, the extracted scene edges that are inside this region are tested for match candidacy. Let $\hat{z}_j$ be a scene feature inside the search region of $g_{m_i,\mathbf{p_0}}$. We evaluate the Mahalanobis distance measure for the image error between $\hat{z}_j$ and $g_{m_i,\mathbf{p_0}}$ as defined by the following equation:

$$d_{f(\bar{\mathbf{p}}_\mathbf{0},m_i,\hat{z}_j)} = f(\bar{\mathbf{p}}_\mathbf{0},m_i,\hat{z}_j)^T \Sigma^{-1}_{f(\mathbf{p_0},m_i,z_j)} f(\bar{\mathbf{p}}_\mathbf{0},m_i,\hat{z}_j) \quad (9)$$

where $\Sigma_{f(\mathbf{p_0},m_i,z_j)}$ is the covariance matrix of $f(\mathbf{p_0},m_i,\hat{z}_j)$, which is calculated by the first-order approximation as follows:

$$\Sigma_{f(\mathbf{p_0},m_i,z_j)} = \frac{\partial f(\mathbf{p_0},m_i,z_j)}{\partial \mathbf{p_0}}^T \Sigma_{\mathbf{p_0}} \frac{\partial f(\mathbf{p_0},m_i,z_j)}{\partial \mathbf{p_0}} + V_j. \quad (10)$$

Assuming $f(\bar{\mathbf{p}}_\mathbf{0},m_i,\hat{z}_j)$ is locally Gaussian in the vicinity of $\bar{\mathbf{p}}_0$ and $\hat{z}_j$, the distance measure $d_{f(\bar{\mathbf{p}}_\mathbf{0},m_i,\hat{z}_j)}$ has Chi-squared distribution with 4 DOF, since $f(\bar{\mathbf{p}}_\mathbf{0},m_i,\hat{z}_j)$ is defined to be a 4-D vector. With confidence level of 0.5, we choose z_j as a match candidate for $g_{m_i,\mathbf{p_0}}$ if $d_{f(\bar{\mathbf{p}}_\mathbf{0},m_i,\hat{z}_j)}$ is less than $\chi_{4,0.50}$, which is 3.357.

The match candidates for each projected model feature in the expectation map constitute a set of model and scene feature correspondences. For convenience of notation, we denote such a set by C_0 in the rest of this paper.

E. Estimating Model and Scene Feature Correspondence Using Hypothesis Generation and Verification Scheme

After the set of model and scene feature correspondences is constructed, we determine the true correspondences, these being correspondences that satisfy certain criteria that we will present in the following subsections. A matching hypothesis, which we denote as C_H for convenience of notation, is a subset of the initial feature correspondence set C_0. For selecting C_H, we use a priority selection scheme that uses a certain weight measure for each feature correspondence pair in C_0. The weight measure is calculated with three heuristic rules that are described in the next subsection. While selecting feature correspondences for C_H, the feature pair that has a higher weight measure is given higher priority. Each time a feature correspondence pair is selected for a match hypothesis, the predicted pose $\mathbf{p_0}$ is updated with the feature pair. Hence, such updated pose, which we denote as $\mathbf{p_H}$, represents the best estimate of the object pose for the feature correspondences that are currently selected for C_H. The feature correspondence selection procedure continues until the pose uncertainty associated with $\mathbf{p_H}$ is reduced below a certain threshold.

After C_H is constructed, it is verified with the two criteria that are described in Section III-G. If C_H is rejected, then the system regenerates C_H based on the criterion violated. Such regeneration procedures are also described in Section III-G. The hypothesis generation and verification process iterates until the two verification criteria are all satisfied, or no more model and scene feature pairs are available for generating hypotheses.

In Fig. 6, the overall control flow of the hypothesis generation and verification scheme is shown. In the next subsection, we describe the details of how the matching hypothesis C_H is generated.

F. Hypothesis Generation

Regarding the number of feature correspondences needed for the hypothesis, one widely accepted strategy, such as the one with the random sample consensus (RANSAC) approach [25], is to use only the minimum required number of feature correspondences that guarantee a certain level of confidence in the estimated pose, and then, to verify the hypothesized feature correspondences with the estimated pose. The pose uncertainty associated with the estimated pose translates directly into the confidence level. Since we estimate the pose by minimizing the error between the projection of the model and the corresponding scene edges in the image space, we must also calculate the pose uncertainty in the image space. This requires that we project the 6 DOF uncertainty into the image plane.

In our EKF-based pose update framework, the extent to which each pose update reduces the uncertainty associated with the

pose is controlled by the Kalman gain that is subject to the measurement uncertainty for the scene features, as shown in (8). Since the measurement uncertainty for scene features is subject to the image noise and the errors in the straight-edge detection process, it cannot be predicted in advance as to how much uncertainty would be reduced by a single iteration of pose update. We therefore update the pose incrementally for each matched pair of model and scene edge. At the same time, we compute the new uncertainty associated with the updated pose and project the uncertainty into the image plane. This iterative process stops either when we run out of all model edges, or when the updated pose uncertainty drops below a certain threshold, whichever comes first. The threshold for the projected pose uncertainty is chosen considering the level of pose estimation accuracy required for our application.

For selecting the model and scene feature correspondences for the hypothesized correspondence set C_H, if we can give higher priority to the feature pairs that have higher chance to be correct matches, we can minimize the number of hypotheses that should be generated and verified until we get the correct feature correspondences. For such priority assignment, we calculate a weight measure for each model and scene feature correspondences using three heuristic rules. The heuristic rules are:

1) Give high priority to a model feature that has a small number of matching candidates.
2) Give high priority to a model feature and scene feature pair if the Mahalanobis distance measure between these two features is small.
3) Give high priority to a model feature that is distant from its neighboring model features in the expectation map.

The first heuristic rule is to give a higher weight to a model and scene feature pair if the model feature of the pair has fewer scene candidates than the other model features. Obviously, if a certain projected model feature has many matching candidates, then it is confusing to decide which of the candidates the model feature must be matched with. Hence, the model features with smaller number of candidates have higher chance to be correctly matched.

The second heuristic rule is to give a higher weight to a feature pair m_i, z_j if it has a smaller value for the distance measure $d_{f(\bar{\mathbf{p}}_\mathbf{0}, m_i, \hat{z}_j)}$ that is presented in (9). Although the predicted pose $\mathbf{p_0}$ is likely to contain errors, as we mentioned previously, it remains that $\mathbf{p_0}$ is our current best estimate the object pose. For this reason, we assume that any feature pair that has a small image error with regard to the current best estimate of the object pose has a higher chance to be a correct match.

The third heuristic rule is chosen based on the observation that a model feature whose projection in the expectation map is geometrically distant—both in location and in direction—to the other projected model features is less likely to be mismatched. If two model features are located close to each other in the image space, significant parts of their search regions may overlap. Hence, the chance of two different similarly shaped candidate scene features to be in both search regions would be high, making it more difficult to choose correct matches for the two model features. For example, as shown in Fig. 7, there is significant overlap between the search regions of the two projected model features labeled $MF1$ and $MF2$. Note here that there exist multiple scene features that have similar lengths and orientations in both search regions.

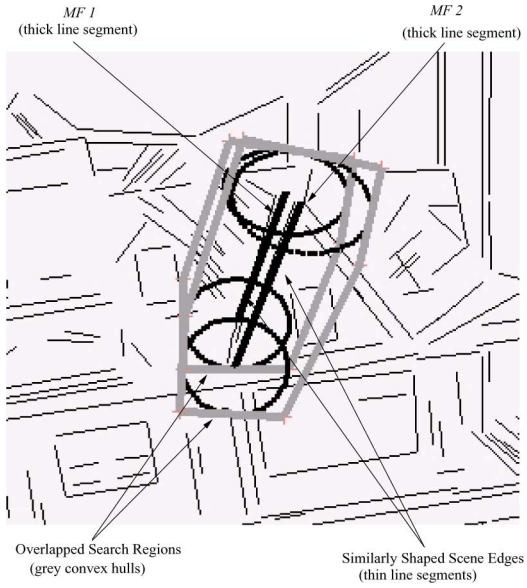


Fig. 7. Search regions for two different model lines overlapping significantly. Note that the search region for a model line is the convex hull of the two projected uncertainty regions for the two line extremities.

There have been previous approaches, including the one by Tonko and Nagel [15], that disregard model features that are geometrically close to each other in generating the expectation map. Our approach is different from those approaches in the sense that such geometrically close model features are included in the expectation map with low priority instead of being completely disregarded. With this strategy, the matches for the low-priority features are sought when the higher priority features fail to match, hence increasing the level of fault tolerance.

In order to use the third heuristic rule for calculating the matching weight, we define for each projected model feature $g_{m_i,\mathbf{p_0}}$ the distance measure $d_\varsigma(g_{m_i,\mathbf{p_0}})$ that describes how distant $g_{m_i,\mathbf{p_0}}$ is to other projected model features in the expectation map. The definition of $d_\varsigma(g_{m_i,\mathbf{p_0}})$ is as follows:

$$d_\varsigma(g_{m_i,\mathbf{p_0}}) = \min_{\forall g_{m_q,\mathbf{p_0}} \in G} g_\varsigma^T \Sigma_\varsigma^{-1} g_\varsigma \tag{11}$$

where G is the set of projected model features in the expectation map, $g_\varsigma = g_{m_i,\mathbf{p_0}} - g_{m_q,\mathbf{p_0}}$, and $\Sigma_\varsigma = \Sigma_{g_{m_i,\mathbf{p_0}}} + \Sigma_{g_{m_q,\mathbf{p_0}}}$.

With the three heuristic rules listed earlier, we calculate the weight measure $W(\mathbf{p_0}, m_i, \hat{z}_j)$ for each model feature and scene feature pair (m_i, z_j) as follows:

$$W(\mathbf{p_0}, m_i, \hat{z}_j) = \frac{d_\varsigma(g_{m_i,\mathbf{p_0}})}{d_{f(\mathbf{p_0}, m_i, \hat{z}_j)} n_{\mathrm{cand}}(g_{m_i,\mathbf{p_0}})} \tag{12}$$

where $d_{f(\mathbf{p_0}, m_i, \hat{z}_j)}$ is the Mahalanobis distance of the image error between $g_{m_i,\mathbf{p_0}}$ and z_j as defined in (9) and $n_{\mathrm{cand}}(g_{m_i,\mathbf{p_0}})$ is the number of match candidates for $g_{m_i,\mathbf{p_0}}$. For all members of C, we evaluate this weight measure before we start selecting the correspondence pairs for C_H. We then sequentially choose the feature pairs from C as sorted by the values of

the composite weight $W(\mathbf{p_0}, m_i, \hat{z}_j)$. Then, as we mentioned earlier, the current best estimate of pose $\mathbf{p_H}$ with regards to the current hypothesis set C_H is recursively updated with the newly added feature pair. This selection procedure iterates until the uncertainty associated with the updated pose $\mathbf{p_H}$ falls below a certain threshold.

G. Verifying Hypotheses

In the following two sections, we describe the details of the two criteria we use for verifying the hypothesis set C_H and how we backtrack over this set (by examining its subsets) when the criteria are violated.

1) Hypothesis Verification and Modification With the Matching Consensus Criterion: When a hypothesized feature correspondence set C_H is generated from the scene image, we evaluate the sum of the squared image plane errors between the model features and the scene features in the set C_H as follows:

$$d_{C_H,\mathbf{p_H}} = \sum_{i=1}^{|C_H|} d_{f(\mathbf{p_H}, m_{H_i}, \hat{z}_{j_{H_i}})} \tag{13}$$

where the term $d_{f(\mathbf{p_H}, m_{H_i}, \hat{z}_{j_{H_i}})}$ is the image error between a particular model feature m_{H_i} and its corresponding scene feature $z_{j_{H_i}}$, as defined in (9) for the case of initial pose. $|C_H|$ denotes the cardinality of C_H. As we have previously mentioned, $d_{f(\mathbf{p_H}, m_{H_i}, \hat{z}_{j_{H_i}})}$ has a Chi-squared distribution with 4 DOF. Hence, $d_{C_H,\mathbf{p_H}}$ also has a Chi-squared distribution with DOFs equal to four times the cardinality of C_H. With the confidence level of 95%, we reject the hypothesis C_H if $d_{C_H,\mathbf{p_H}}$ is greater than $\chi_{4|C_H|,0.95}$.

In order to explain why this criterion is used for verifying C_H, we use the notion of the matching consensus. If the feature correspondence pairs in C_H are true matches, then all the matching pairs must be consistent with a certain estimate of the object pose. In other words, the scene features of the correspondence pairs in C_H must have reasonably small image errors with the corresponding model features projected with the true estimate of the object pose. In that sense, for the hypothesized feature correspondence set C_H to be accepted, the feature correspondence pairs in C_H should form a consensus set with regard to the updated pose $\mathbf{p_H}$. For this reason, we call this criterion as the matching consensus criterion.

When the matching consensus test fails for a hypothesis C_H, there are one or more feature correspondence pairs in C_H that are not consistent with the pose hypothesis $\mathbf{p_H}$; these result in large image errors.

In order to remove these inconsistent correspondences from the hypothesis C_H, we use the following "leave one out" approach to detect the model-scene feature pairing that is most inconsistent with the rest of the pairings. This is done by applying the matching consensus criterion to each of the subsets of C_H in the following manner.[3]

1) For each model and scene feature pair $(m_{H_i}, z_{j_{H_i}})$ in C_H, we make a subset C_{H_i}, which is defined as follows:

$$C_{H_i} = C_H - \{(m_{H_i}, z_{j_{H_i}})\}. \tag{14}$$

2) For each subset C_{H_i}, we update the predicted pose $\mathbf{p_0}$. Obviously, this pose calculation only uses the model-scene feature pairings in C_{H_i}. That is, the new updated pose would *not* include the feature pair $(m_{H_i}, z_{j_{H_i}})$. The new updated pose is called $\mathbf{p_{H_i}}$.
3) We evaluate the matching consensus criterion for each subset C_{H_i} with $\mathbf{p_{H_i}}$. Let $C_{H_{\min}}$ be the subset with the minimum matching consensus criterion value. The feature pair $(m_{H_{\min}}, z_{j_{H_{\min}}})$ that corresponds to $C_{H_{\min}}$ is chosen as an inconsistent correspondence pair. $C_{H_{\min}}$ constitutes the modified hypothesis after removing the inconsistent feature pair from C_H.

This approach is based on the assumption that the inconsistent correspondences do not form a consensus set by themselves. Hence, if an inconsistent correspondence is removed from the hypothesis set C_H, the updated pose with the new hypothesis set is closer to the pose for the consensus subset in the hypothesis.

If C_H includes more than one inconsistent correspondence pair, the new hypothesis set $C_{H_{\min}}$ may not satisfy the matching consensus criterion. In this case, we execute again the inconsistent pair detection procedure described earlier until the modified hypothesis subset satisfies the matching consensus criterion.

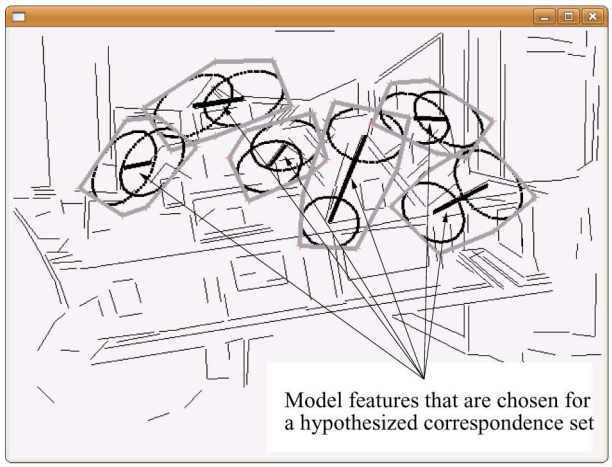
After we find the subset of C_H that satisfies the matching consensus criterion, the system uses the hypothesis generation algorithm that was described in Section III-F to add more feature correspondences to C_H, and verifies the modified hypothesis set again with the matching consensus criterion.

2) Assigning Nil-Mappings and Verifying the Hypothesis Based on the Number of Nil-Maps: If C_H satisfies the matching consensus criterion, the model features that are not included in C_H are projected into the image with pose $\mathbf{p_H}$, and the matching candidates for such projected model features are sought.

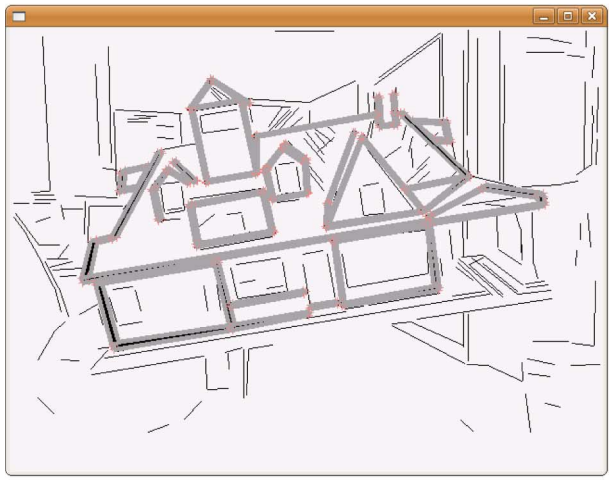
Since an accepted C_H guarantees a certain bound on the projected uncertainty in the image plane, the remaining model features projected with $\mathbf{p_H}$ have small search regions. For example, Fig. 8(b) shows the search regions for the remaining model features when projected into the camera image with the pose updated with an accepted C_H. Fig. 8(a) displays the model-scene feature pairings in an accepted C_H. Ordinarily, on account of the tight bounds on the uncertainties associated with the projected model features at this point, there will exist at most a single candidate scene feature within the uncertainty region associated with any remaining model feature. If that is the case, that scene feature is chosen for matching with the model feature. If multiple scene features are found in this uncertainty region, the system selects the closest scene feature. And, if no scene feature is inside the uncertainty region, the model feature is assigned a nil-map.

We place a constraint on how many nil-maps are allowed for a given set of model features. It is entirely possible that a C_H was accepted for reasons of accidental alignment between a partial region of the model with a partial region of the scene without

[3]It would seem that, in the worst case, this would require computations that depend exponentially on the size of C_H. But note that it takes a small number of features, of the order of unity, to estimate the pose of a rigid object. So, C_H will contain only a small number of feature pairings, typically five or six.



(a)



(b)

Fig. 8. Search regions for the projected model features before and after the pose update using a hypothesized feature correspondence set. (a) Search regions for the model lines using the hypothesized correspondence set before the pose update. Each search region is a convex hull of the projected uncertainties for the line extremities. (b) Search regions for the model lines after a pose update. These regions so closely enclose the model lines that they are indistinguishable from the model lines.

the object and its image being in true global alignment. So, if the number of nil-maps exceeds a threshold, we reject the entire C_H and regenerate another C_H. When a C_H is rejected on account of too many nil-maps, the model-to-scene correspondence in the rejected C_H is not allowed to occur again. For that reason, a rejected C_H is guaranteed not to appear again.

IV. Scene Feature Extraction Module and Feature Measurement Uncertainty

A. Extracting Straight-Line Edges

For extracting straight-line edges from the input scenes, we first calculate a gradient edge map with the Canny operator [26]. Straight-line edges are then extracted by grouping the edge fragments in the gradient edge map using the grouping algorithm presented in [27].

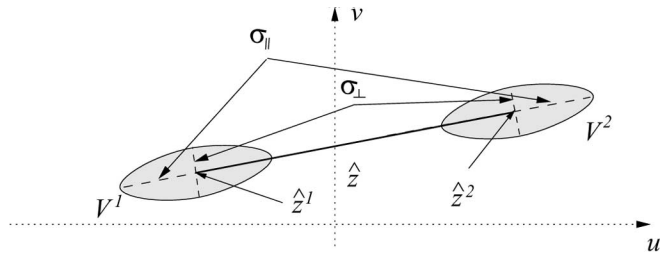


Fig. 9. Scene edge measurement uncertainty.

This grouping algorithm runs much faster than the popular Hough transform algorithm. Also, the algorithm directly produces the 2-D Cartesian coordinates of the straight-edge extremities unlike the Hough transform algorithm, which needs postprocessing for calculating the image coordinates of the end points.

B. Calculating the Measurement Uncertainty

Let $\hat{z}$ be the measured version of a straight-line scene feature z. We denote the noise uncertainty associated with such measurement as a 4×4 matrix V. Assuming that the location measurements of the two end points of z are independent of each other, we denote V as follows:

$$V = \begin{pmatrix} V^1 & 0 \\ 0 & V^2 \end{pmatrix} \tag{15}$$

where V^k, $k = 1, 2$, are 2×2 covariance matrices that correspond to the image coordinates of the two end points of z. Since we assume each end point of z is a 2-D Gaussian variable, V^k is represented with two uncertainty spans denoted as $\sigma_\perp$ and $\sigma_\parallel$, as shown in Fig. 9. $\sigma_\perp$ and $\sigma_\parallel$ represent the uncertainties associated with the measurement noise in perpendicular direction and parallel direction to $\hat{z}$, respectively.

If we know the estimated value for $\sigma_\parallel$ and $\sigma_\perp$, V^k is calculated as follows [28]:

$$V^1 = V^2 = \begin{pmatrix} \sigma_\parallel^2 \cos^2\theta + \sigma_\perp^2 \sin^2\theta & \left(\sigma_\parallel^2 - \sigma_\perp^2\right)\cos\theta\sin\theta \\ \left(\sigma_\parallel^2 - \sigma_\perp^2\right)\cos\theta\sin\theta & \sigma_\parallel^2 \sin^2\theta + \sigma_\perp^2 \cos^2\theta \end{pmatrix}. \tag{16}$$

1) Calculating $\sigma_\perp$: $\sigma_\perp$ for each straight line is calculated in a bottom-up fashion as follows.

For extracting straight line segments, edge pixels are first grouped into edge clusters that are consecutively aligned in horizontal or vertical directions. Such clusters of edge pixels are called "linear-primitives." When we group edge pixels into linear-primitives, the locational uncertainties for the end points of each linear-primitive are assigned values proportional to the variance associated with Canny edge detection. Later, when we group linear-primitives into a straight-line, we identify the linear-primitive end point, labeled as p_f in Fig. 10(a), that is farthest from the straight line fitted to the linear-primitives. $\sigma_\perp$ for the straight line is calculated as the sum of the distance from

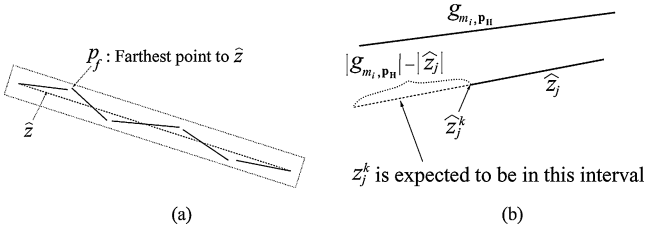


Fig. 10. Estimating edge measurement noise uncertainty. (a) Estimating $\sigma_\perp$ (b) Estimating $\sigma_\parallel$.

p_f to the straight line and the locational uncertainty assigned for p_f.

2) Calculating $\sigma_\parallel$: $\sigma_\parallel$ is calculated using the following equation:

$$\sigma_\parallel = \sigma_{\text{frag}} + \sigma_{\text{location}} \tag{17}$$

where σ_{frag} is the uncertainty associated with the measurement noise induced by edge fragmentation, and σ_{location} is the locational uncertainty associated with the two end points of $\hat{z}$. σ_{location} is the end point location uncertainty assigned to the linear-primitive to which the end point corresponds. As already stated, this uncertainty is made proportional to the variance associated with Canny edge detection. The problem is that it is difficult, if not impossible, to analytically estimate σ_{frag} by locally observing the scene features. However, as we will show later, empirical estimates of σ_{frag} can be generated if we have available to us the expected length of the true edge in the scene. Fortunately, this expected length is available to us in two out of three situations where we need σ_{frag}. In the remaining case, we must make do by assuming a reasonable constant value for the expected length.

The three usage situations where σ_{frag} is needed are as follows:

1) Evaluating the Mahalanobis distance $d_{f(\bar{\mathbf{p}}_\mathbf{0},m_i,\hat{z})}$ in (9) when we search for the matching candidates for a projected model feature.
2) Updating the pose with a given correspondence between a projected model feature $g_{m_i,\mathbf{p_H}}$ and the corresponding scene feature $\hat{z}$.
3) Evaluating the matching consensus criterion in (13) with given hypothesized correspondences between the projected model features $g_{m_i,\mathbf{p_H}}$ and the scene features $\hat{z}$.

For the first situation, the expected length of the true edge cannot be calculated. In this case, we use a constant value that is chosen empirically. For the second and third situations, we can assume the expected length of the true edge z to be the length of the corresponding projected model edge $g_{m_i,\mathbf{p_H}}$. In this case, as shown in Fig. 10(b), an end point of the true edge z^k is expected to be on the extended line emanating from measured end point $\hat{z}^k$ within the range of the difference of the lengths of $g_{m_i,\mathbf{p_H}}$ and $\hat{z}$. Assuming the probability of the end points $z^k, k = 1, 2$ to be located in this range is uniformly distributed, the expected value of the fragmentation error is $(|g_{m_i,\mathbf{p_H}}| - |\hat{z}|)/2$, where $|g_{m_i,\mathbf{p_H}}|$ denotes the length of the projected model line and $|\hat{z}|$ the length of the scene line. We use this value as the estimate of σ_{frag}.

V. Pose Prediction Module

For each image frame, the initial statistics of the pose of the target object are predicted using the history of the estimates of the object pose for the previous frames.

Let $\mathbf{p^k}$ be the pose estimate at time t^k with k being the time index. We denote the mean and covariance of $\mathbf{p^k}$ with $\bar{\mathbf{p}}^\mathbf{k}$ and $\Sigma_{\mathbf{p^k}}$, respectively. Then, the initial pose for the next time stamp t^{k+1}, which we denote $\mathbf{p^{k+1}}$, is given by

$$\mathbf{p^{k+1}} = \left(1 + \frac{\delta t^{k+1}}{\delta t^k}\right)\mathbf{p^k} - \frac{\delta t^{k+1}}{\delta t^k}\mathbf{p^{k-1}} + \zeta^{k+1} \tag{18}$$

where $\delta t^k = t^k - t^{k-1}$ and where ζ^{k+1} denotes the prediction noise with zero mean and covariance equal to $\Sigma_{\zeta^{k+1}}$.

Assuming the prediction noise ζ^{k+1} is independent of the pose estimates, the predicted covariance of $\mathbf{p^{k+1}}$ is calculated as follows.[4]

$$\Sigma_{\mathbf{p^{k+1}}} = \left(1 + \frac{\delta t^{k+1}}{\delta t^k}\right)^2 \Sigma_{\mathbf{p^k}} + \left(\frac{\delta t^{k+1}}{\delta t^k}\right)^2 \Sigma_{\mathbf{p^{k-1}}} + \Sigma_{\zeta^{k+1}}. \tag{19}$$

With our pose prediction model, the magnitude of the prediction error depends on the magnitude of the motion jitter of the target object. When the motion jitter is bounded—for example, when the target object is hanging on a gantry and the motion jitter mainly comes from the inertial forces—we estimate $\Sigma_{\zeta^{k+1}}$ from an image sequence captured for a certain time interval. However, if the motion jitter does not have a bound—for example, if a human user shakes the target object at his/her will—then, it is impossible to correctly estimate $\Sigma_{\zeta^{k+1}}$. In this case, we assume an arbitrarily large value for $\Sigma_{\zeta^{k+1}}$.

VI. Experimental Results

If the reader would be willing to indulge us, to best experience our experimental results, he/she is asked to point his/her browser to the web site http://cobweb.ecn.purdue.edu/RVL/Projects/ModelBasedTracking/index.htm.

There, the reader will see a human shaking an object as it is being tracked in real time. Another demonstration at that site shows successful tracking even when the object is significantly occluded by a human waving his hand between the camera and the object.[5]

In all of the experiments we present in this section, we must provide the tracker with the initial pose of the target object. How this initial pose information is supplied is different for different types of experiments. For the visual-servoing-based assembly-on-the-fly experiments, the initial pose is provided by the "coarse module" as described in [19]. The coarse module uses a ceiling-mounted camera for rough estimation of the

[4]With the Markovian assumption, the previous two estimates $\mathbf{p^k}$ and $\mathbf{p^{k-1}}$ are not independent. We ignore the correlation term of $\mathbf{p^k}$ and $\mathbf{p^{k-1}}$ for calculating the predicted covariance, because the uncertainty associated with these estimates is very small after pose estimation with the Kalman filter.

[5]Movies that demonstrate the experiments described in this section are also available for download at http://ieeexplore.ieee.org.

location of the target. When the target moves into the servo range of the robot end-effector-mounted camera, the control is automatically handed over to the "fine control" module that is also described in [19]. The "fine control" module is based on the tracking algorithms described in this paper. For the tracking of handheld objects in real time, we have developed a GUI that gives the user control over translations and rotations of the wireframe model as projected onto a terminal screen. The user can manually bring the model into correspondence with the first camera image and thus initialize the pose. A similar GUI-based approach is used for pose initialization when tracking an object in video sequences offline.

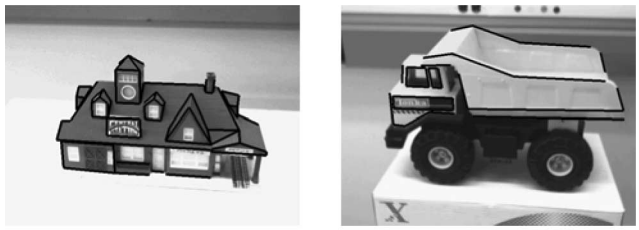
In the rest of this section, we will first present quantitative results on the accuracy of tracking for two different objects. For each object, we analyzed two video sequences for estimating tracking accuracy. Subsequently, we will show qualitative results on these two objects and two additional objects possessing complex shapes.

A. Quantitative Analysis of Pose Estimation Errors

As stated in the preamble to this section, we report our tracking accuracy results with the help of two video sequences of two different objects, the train-station object and a truck object. We will refer to the two video sequences of the train-station object as *Station-Smooth* and *Station-Nonsmooth*. Similarly, we will refer to the two video sequences of the truck object as *Truck-Smooth* and *Truck-Nonsmooth*. The "smooth" and "nonsmooth" qualifiers in the names reflect the nature of the motion of the object with respect to the camera. In particular, the motion for the "nonsmooth" case is very jerky as should be evident to those visiting the URL mentioned at the beginning of this section. For the "nonsmooth" case, the results shown later also include plots of the translational and rotational parameters as functions of time to give the reader an idea of the jerkiness of the motions. The overall size of the train station object is $175 \times 100 \times 58$ and that of the truck $410 \times 250 \times 200$, all in millimeters.

For calculating the ground truth pose for these image sequences, we mount the camera on a high-performance robotic arm, and we move the robotic arm while keeping the target object stationary. Since the calculation of the pose of the object is always relative to the coordinate frame of the cameras, moving the cameras while the object is stationary is equivalent to tracking a moving object with a stationary camera. With this approach, the ground truth pose is calculated from the robot kinematics and the relative pose of the camera with respect to the robot end-effector, which is given by hand–eye calibration.

For the *Station-Smooth* and *Truck-Smooth* experiments, the camera mounted on the robot end-effector moved along a designated path. The average distance from the camera to the target object was 350 mm for the *Station-Smooth* sequence and 700 mm for the *Truck-Smooth* sequence. Each video sequence contained 100 images. A frame from each of the two sequences is shown in the composite in Fig. 11. Shown below the images are two sets of numbers. The first set is the true pose of the object in the camera coordinate frame and the second set is the estimated pose. The average rms error for *Station-Smooth* sequence is 3.3 mm in translation and 0.27° in rotation, and for *Truck-Smooth* sequence it is 6.2 mm in translation and 0.20° in rotation.



Station-Smooth					
t_x	t_y	t_z	ϕ_x	ϕ_y	ϕ_z
2.2,	51.4,	361.7,	-102.4,	-65.6,	-75.1
1.6,	50.5,	362.5,	-100.8,	-64.9,	-76.6

Truck-Smooth					
t_x	t_y	t_z	ϕ_x	ϕ_y	ϕ_z
-59.3,	-129.6,	701.1,	25.9,	13.1,	10.2
-63.5,	-132.2,	701.3,	25.9,	13.1,	10.0

Fig. 11. Accuracy analysis results for the train-station object and the truck object. For each image, the first six numbers denote the pose values of the ground truth and the second six numbers denote the corresponding pose estimates.

For the other two image sequences, *Station-Nonsmooth* and *Truck-Nonsmooth*, the robot end-effector with the camera mounted on it was made to execute sudden large random changes in its direction. In order to give the reader a sense of the magnitude of the motion jerkiness thus induced between the camera and the objects, Fig. 12 shows the translation and rotation values of the ground truth pose (solid line with circular markers) and the corresponding estimates for the tracked pose (dashed line with cross markers) for the two sequences. The horizontal axis in all the plots shown is the time in the video sequences.

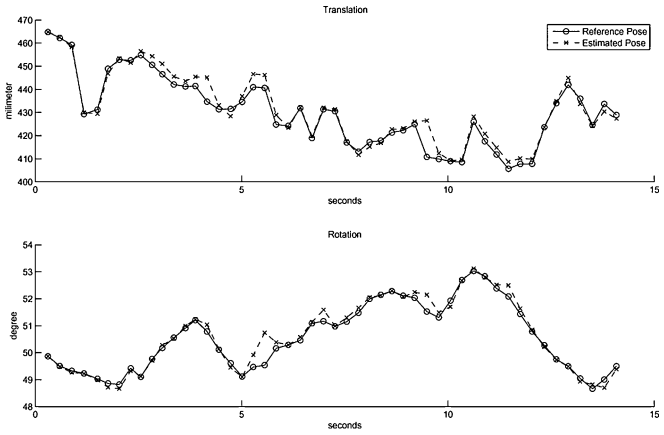
To give the reader an even better sense of the extent of motion jerkiness injected manually into the two Nonsmooth experiments, Fig. 13 shows a frame from each video sequence. For each frame, we show the object as it appears to the camera, its superimposed predicted pose with gray line segments, and its estimated pose with dark line segments.

As shown in Table I, the average rms error for the *Station-Nonsmooth* sequence is 4.8 mm in translation and 0.36° in rotation, and 9.2 mm and 0.67° for the *Truck-Nonsmooth* sequence. The table also includes entries for the average rms error for the case of smooth motions.

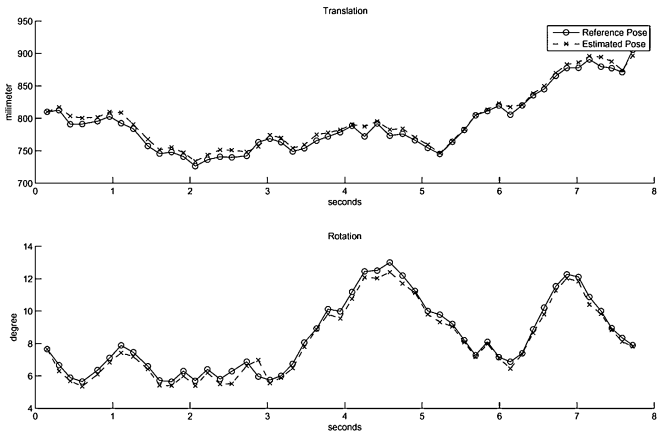
B. Pose Estimation Performance Analysis

While the previous section reported quantitative results on the tracking accuracy, we will now address the issue of tracking performance, meaning the speed with which the objects can be tracked. The performance numbers will be presented for the same four video sequences used in the previous section. The computer hardware used in those tracking experiments was a Pentium-4 3.6 GHz processor with 512 Mb of system memory.

Obviously, the time it takes to update the object pose depends on the number of features in the model for the target object. The number of physical edges used for the train station model was 148 and for the truck model 67. A significant portion of the processing time is spent on low-level image processing such as smoothing, edge detection, and straight-edge extraction. The mean of the scene feature extraction time was 153 ms for the

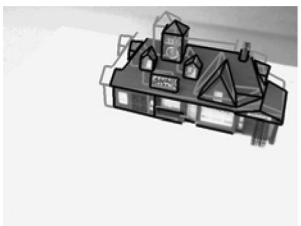


(a)



(b)

Fig. 12. (a) Vertical axis in the top panel represents the magnitude of the true translational displacement and its estimated value for the case of nonsmooth motion for the station object. The vertical axis in the lower panel is magnitude of the true rotational displacement and its estimate for the same object. The horizontal axis is in seconds. (b) Same as in (a) but for the truck object.



Station-Nonsmooth

t_x	t_y	t_z	ϕ_x	ϕ_y	ϕ_z
45.9,	-3.0,	423.5,	-111.2,	-47.3,	-61.2
40.9,	1.5,	426.2,	-109.6,	-47.8,	-62.4

Truck-Nonsmooth

t_x	t_y	t_z	ϕ_x	ϕ_y	ϕ_z
-113.7,	-63.5,	883.5,	30.5,	12.5,	9.6
-110.8,	-64.9,	887.4,	28.6,	13.6,	10.6

Fig. 13. Accuracy analysis results for *Station-Nonsmooth* and *Truck-Nonsmooth* sequences.

train station object and 151 ms for the truck object. The pose estimation time obviously depends on the number of EKF iterations to achieve convergence in the matching of scene features and model features. The average number of EKF iterations is seven for the *Station* sequences and six for the *Truck* sequences. The mean of the pose estimation time was only 53 ms for the train station object and 10 ms for the truck object for the four image sequences used in the previous section. Processing time for each of the four image sequences is listed in Table I.

TABLE I
POSE ESTIMATION ERROR AND PROCESSING TIME
FE: feature extraction; PE: pose estimation

Image Sequence	RMS Error		Processing Time	
	Trans. (mm)	Rot. (deg)	FE (ms)	PE (ms)
Station-Smooth	3.3	0.27	173	53
Station-Nonsmooth	4.8	0.36	132	53
Truck-Smooth	6.2	0.20	156	10
Truck-Nonsmooth	9.2	0.67	147	10

C. Some Further Tracking Experiments in the Presence of Occlusion and Cluttered Background

We will now present some additional experimental results to demonstrate how robust our tracking method is to occlusion and cluttered backgrounds. Since each of these phenomena is difficult to quantify for nontrivial experimental conditions, our results in the rest of the paper are only qualitative. That is, we will show some example frames from experimental data taken under conditions that represent the phenomena. With overlays, these example frames will demonstrate that our system is able to track a target object despite the presence of highly adverse circumstances. But, for obvious reasons, it is difficult to convey the full sense of the capabilities of our approach using just static images. An interested reader is therefore urged to visit the Web site whose URL was mentioned at the beginning of Section VI.

For each of the two objects used in the previous section, the train-station object and the truck object, we captured one image sequence with highly cluttered background, named *Station-Clutter* for the train-station object and *Truck-Clutter* for the truck object, and one image sequence in the presence of severe occlusion, named *Station-Occlusion* and *Truck-Occlusion*, respectively.

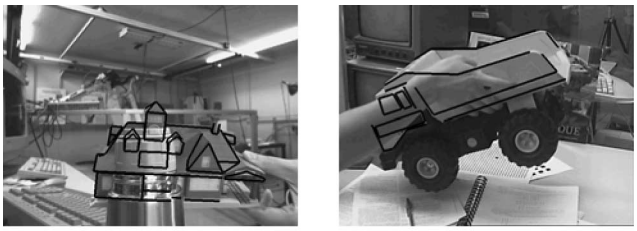
A frame from each of the two video sequences *Station-Clutter* and *Truck-Clutter*, presented in Fig. 14, qualitatively demonstrates the robustness of our technique when the background is highly cluttered. Superimposed on each frame is projection of the model into the camera image using the calculated pose of the object.

Along the same lines, a frame for each of the two video sequences, *Station-Occlusion* and *Truck-Occlusion*, in Fig. 15 demonstrates the robustness of the system with regard to heavy occlusion. The superimposed wireframe (thick black line segments) in each image shows that the object is being tracked correctly despite the fact that a significant portion of the object is occluded.

We also tested our algorithm for tracking two visually challenging objects. A jeep object, shown in the left panel of Fig. 16, has a complicated shape making the feature matching process confusing. The other object is a digital camera, shown in the right panel of Fig. 16, which has metallic and dark surfaces that make it difficult to extract scene features under normal

Station-Clutter *Truck-Clutter*

Fig. 14. Sample frames of *Station-Clutter* and *Truck-Clutter* sequences.



Station-Occlusion *Truck-Occlusion*

Fig. 15. Sample frames of *Station-Occlusion* and *Truck-Occlusion* sequences.

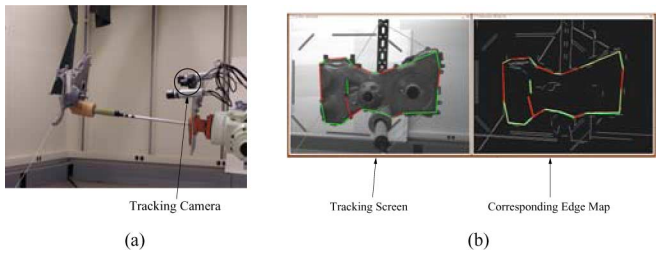
Jeep *Camera*

Fig. 16. Sample frames of *jeep* and *camera* sequences.

lighting conditions. Experiments show that the system successfully tracks these two objects. Complete tracking sequences for these objects are also posted at the URL mentioned earlier.

D. Visual Servoing Experiment

Finally, we will present an experiment in real-time visual servoing using the object tracking algorithm presented in this paper. The goal of this experiment is to carry out peg-in-hole assembly while the "hole" is undergoing large and nonsmooth motions. Fig. 17(a) shows an engine-cover object that hangs from a gantry mounted on the ceiling. The object contains a "hole" into which the robot must insert a "peg." In a typical experiment, the engine-cover object moves along a linear slide with an average speed of 43.5 mm/s. A couple of strings are attached to the engine cover so that a human can pull them differentially to induce large jerkiness in the motion of the "hole" as the robot end-effector tries to insert the peg into it. Fig. 17(a) shows an example of successful peg insertion, and a screen shot of the camera images before and after feature extraction for successful insertion is shown in Fig. 17(b). Further details regarding these visual servoing experiments for the purpose of robotic assembly are presented in [19]. The servoing results can also be seen



(a) (b)

Fig. 17. Our real-time visual servoing system. Note that the stereo camera shown in (a) is used by another visual tracking module that is a part of a multiple-vision-loop architecture. Detailed description of the architecture is presented in [19].

at http://cobweb.ecn.purdue.edu/RVL/movies/LineTracking/ICRA06.wmv.

VII. Conclusion

In this paper, we presented a model-based visual tracking system that gives 3-D pose estimates of a rigid object as it is tracked with a single camera. Our system can accurately estimate the pose of a moving object while showing robustness in the presence of severe occlusion, highly cluttered background, and nonsmooth motion of the object.

Our research employed straight-line object features exclusively for the purpose of object tracking. But our core tracking framework could be used with any set of object features provided they can be extracted reliably and without excessive computational delays.

Acknowledgment

The authors are grateful to F. Maslar of the Advanced Manufacturing Technology Development (AMTD), Ford Motor Company, for being a critical driving force behind the research reported here. This research would not have come about without his constant and constructive feedback. They also thank J. Park for his model building expertise and G. DeSouza for helpful discussions regarding Kalman filtering.

References

[1] T. N. Tan, G. D. Sullivan, and K. D. Baker, "Pose determination and recognition of vehicles in traffic scenes," in *Proc. Eur. Conf. Comput. Vision*, Stockholm, Sweden, May 1994, pp. 501–506.

[2] D. Koller, K. Daniilidis, and H. H. Nagel, "Model-based object tracking in monocular image sequences of road-traffic scenes," *Int. J. Comput. Vision*, vol. 10, no. 3, pp. 257–281, 1993.

[3] W. F. Gardner and D. T. Lawton, "Interactive model-based vehicle tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 18, no. 11, pp. 1115–1121, Nov. 1996.

[4] A. E. C. Pece and A. D. Worrall, "Tracking with the EM contour algorithm," in *Proc. Eur. Conf. Comput. Vision*, 2002, pp. 3–17.

[5] R. Sharma and J. Molineros, "Role of computer vision in augmented virtual reality," *The SPIE Conf. on the Engineering Reality of Virtual Reality*, Feb. 1995, San Jose, CA.

[6] A. Kosaka and A. C. Kak, "Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties," *Comput. Vision Graph. Image Process.: Image Understanding*, vol. 56, no. 3, pp. 271–329, 1992.

[7] G. N. DeSouza and A. C. Kak, "A subsumptive, hierarchical, and distributed vision-based architecture for smart robotics," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 34, no. 5, pp. 1988–2002, Oct. 2004.

[8] C. Harris, "Tracking with rigid models," in *Active Vision*, A. Blake and A. Yuille, Eds. Cambridge, MA: MIT Press, 1992, pp. 59–73, ch. 4.
[9] T. Drummond and R. Cipolla, "Real-time visual tracking of complex structures," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 932–946, Jul. 2002.
[10] F. Martin and R. Horaud, "Multiple-camera tracking of rigid objects," *Int. J. Robot. Res.*, vol. 21, no. 2, pp. 97–113, Feb. 2002.
[11] E. Marchand, P. Bouthemy, and F. Chaumette, "A 2D–3D model-based approach to real-time visual tracking," *Image Vision Comput.*, vol. 19, no. 13, pp. 941–955, Nov. 2001.
[12] L. Vacchetti, V. Lepetit, and P. Fua, "Stable real-time 3D tracking using online and offline information," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1385–1391, Oct. 2004.
[13] V. Lippiello, B. Siciliano, and L. Villani, "Position-based visual servoing in industrial multirobot cells using a hybrid camera configuration," *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 73–86, Feb. 2007.
[14] D. G. Lowe, "Robust model-based motion tracking through the integration of search and estimation," *Int. J. Comput. Vision*, vol. 8, no. 2, pp. 113–122, 1992.
[15] M. Tonko and H.-H. Nagel, "Model-based stereo-tracking of non-polyhedral objects for automated disassembly experiments," *Int. J. Comput. Vision*, vol. 37, no. 1, pp. 99–118, 2000.
[16] A. Kosaka and G. Nakazawa, "Vision-based motion tracking of rigid objects using prediction of uncertainties," in *Proc. IEEE Int. Conf. Robot. Autom.*, Nagoya, Japan, 1995, pp. 2637–2644.
[17] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence*. New York: McGraw-Hill, 1987.
[18] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," *J. Appl. Mech.*, vol. 77, pp. 215–221, 1955.
[19] Y. Yoon, J. Park, and A. C. Kak, "A heterogeneous distributed visual servoing system for real-time robotic assembly applications," in *Proc. IEEE Int. Conf. Robot. Autom.*, Orlando, FL, 2006, pp. 4422–4424.
[20] O. D. Faugeras, *Three-Dimensional Computer Vision*. Cambridge, MA: MIT Press, 1993.
[21] Z. Zhang, "A flexible new technique for camera calibration," Microsoft Res., Redmond, WA, Tech. Rep. MSR-TR-98-71, Mar. 1999.
[22] Intel. Open source computer vision library [Online]. Available: http://www.intel.com/technology/computing/opencv/index.htm
[23] G. Welch and G. Bishop, "An introduction to the Kalman filter," Tech. Rep. TR 95-041, Univ. of North Carolina, Chapel Hill, 1995.
[24] M. Paterson and F. Yao, "Efficient binary space partitions for hidden surface removal and solid modeling," *Discrete Comput. Geom.*, vol. 5, no. 5, pp. 279–304, 1990.
[25] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with application to image analysis and automated cartography," *Commun. ACM—Graph. Image Process.*, vol. 24, no. 6, pp. 381–395, 1981.
[26] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, Nov. 1986.
[27] D. Lagunovsky and S. Ablameyko, "Straight-line-based primitive extraction in grey-scale object recognition," *Pattern Recognit. Lett.*, vol. 20, pp. 1005–1014, 1999.
[28] R. Deriche and O. Faugeras, "Tracking line segments," *Image Vision Comput.*, vol. 8, no. 4, pp. 261–270, 1990.

Youngrock Yoon (S'99–M'06) received the M.S. and B.S. degrees from the Department of Computer Science, Yonsei University, Seoul, Korea, in 1998 and 1996, respectively, and the Ph.D. degree from the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, in 2006.

He is currently a Research Scientist at Tandent Vision Science, Inc., San Francisco, CA. He participated in a project for developing a cooperative unmanned aerial vehicle system for autonomous intelligence, surveillance, and reconnaissance missions at the U.S. Air Force Academy. His current research interests include sensor-based robotics, computer vision, object pose estimation, and tracking.

Akio Kosaka (S'86–M'88) received the B.Eng. and M.Eng. degrees from the University of Tokyo, Tokyo, Japan, in 1981 and 1983, respectively, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 1992.

He is currently an Adjunct Associate Professor of electrical and computer engineering at Purdue University. He is also a Chief Research Scientist in the Future Creation Laboratory, Olympus Corporation, Tokyo, Japan. He was engaged in a wide variety of research projects, including compute-aided neurosurgical navigation systems, 6 DOF haptic interfaces, wearable user interfaces, content-based image retrieval systems, and 3-D digital camera systems. His current research interests include computer vision, 3-D medical imaging, distributed sensor networks, intelligent environments, and mobile robot navigation.

Avinash C. Kak is a Professor of electrical and computer engineering at Purdue University, West Lafayette, IN. He has coauthored the books *Principles of Computerized Tomographic Imaging* [republished as a Classic in Applied Mathematics by the Society of Industrial and Applied Mathematics (SIAM)] and *Digital Picture Processing* (Academic, 1982, and considered a classic in computer vision and image processing). His most recent books are a part of his *Objects Trilogy Project*. These include *Programming With Objects* (Wiley, 2003), and *Scripting With Objects* (Wiley, 2008). The third book in the trilogy, *Designing With Objects*, is expected to be out in 2009. His current research interests include wired and wireless camera networks, computer vision, robotics, and high-level computer languages.